(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: 0 569 763 A2

(12) EUROPEAN PATENT APPLICATION

(21) Application number: 93106730.0

(51) Int. Cl.[5]: G06F 15/80

(22) Date of filing: 26.04.93

(30) Priority: 12.05.92 US 881597

(43) Date of publication of application: 18.11.93 Bulletin 93/46

(84) Designated Contracting States: DE FR GB

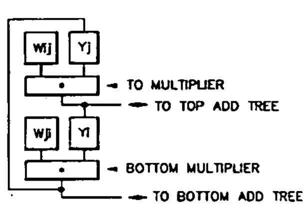
(71) Applicant: INTERNATIONAL BUSINESS MACHINES CORPORATION
Armonk, NY 10504(US)

(72) Inventor: Pechanek, Gerald G.
945 Debonair Drive
Endwell, NY 13760(US)
Inventor: Vassiliadis, Stamatis
717 Vestal Road
Vestal, NY 13850(US)
Inventor: Delgado-Frias, Jose Guadelupe
612 Lacey Drive
Endwell, NY 13760(US)

(74) Representative: Jost, Ottokarl, Dipl.-Ing.
IBM Deutschland Informationssysteme GmbH,
Patentwesen und Urheberrecht
D-70548 Stuttgart (DE)

(54) A massively parallel diagonal fold tree array processor.

(57) A massively parallel processor apparatus having an instruction set architecture for each of the N^2 the PEs of the structure. The apparatus which we prefer will have a PE structure consisting of PEs that contain instruction and data storage units, receive instructions and data, and execute instructions. The N^2 structure should contain "N" communicating ALU trees, "N" programmable root tree processor units, and an arrangement for communicating both instructions, data, and the root tree processor outputs back to the input processing elements by means of the communicating ALU trees. The apparatus can be structured as a bit-serial or word parallel system. The preferred structure contains N^2 PEs, identified as $PE_{column,row,}$ in a N root tree processor system, placed in the form of a N by N processor array that has been folded along the diagonal and made up of diagonal cells and general cells. The Diagonal-Cells are comprised of a single processing element identified as $PE_{i,i}$ of the folded N by N processor array and the General-Cells are comprised of two PEs merged together, identified as $PE_{i,j}$ and $PE_{j,i}$ of the folded N by N processor array. Matrix processing algorithms are discussed followed by a presentation of the Diagonal-Fold Tree Array Processor architecture. The Massively Parallel Diagonal-Fold Tree Array Processor supports completely connected root tree processors through the use of the array of PEs that are interconnected by folded communication ALU trees.



G – "CELLS"

FIG.3B

EP 0 569 763 A2

FIELD OF THE INVENTION

These invention relate to computers and particularly to massively parallel array processors.

REFERENCES USED IN THE DISCUSSION OF THE INVENTION

1. D.E. Rumelhart, J.L. McClelland, and the PDP Research Group, *Parallel Distributed Processing Vol. 1: Foundations* Cambridge, Massachusetts: MIT Press 1986. (Herein referred to as "Rumelhart 86".)
2. E. B. Eichelberger and T. W. Williams, "A Logic Design Structure for Testability," *Proceedings 14th Design Automation Conference, IEEE,* 1977. (Herein referred to as "Eichelberger 77".)
3. J.J. Hopfield, "Neurons With Graded Response Have Collective Computational Properties Like Those of Two-State Neurons," *Proceedings of the National Academy of Sciences 81,* pp. 3088-3092, May, 1984. (Herein referred to as "Hopfield 84".)
4. J.J. Hopfield, "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," *Proceedings of the National Academy of Sciences 79,* pp. 2554-2558, 1982. (Herein referred to as "Hopfield 82".)
5. M.J. Flynn, J.D. Johnson, and S.P. Wakefield, "On Instruction Sets and Their Formats," *IEEE Transactions on Computers Vol. C−34, No. 3,* pp. 242-254, March 1985. (Herein referred to as "Flynn 85".)

In the never ending quest for faster computers, engineers are linking hundreds, and even thousands of low cost microprocessors together in parallel to create super supercomputers that divide in order to conquer complex problems that stump today's machines. Such machines are called massively parallel. Multiple computers operating in parallel have existed for decades. Early parallel machines included the ILLIAC which was started in the 1960s. Other multiple processors include (see a partial summary in U.S. Patent 4,975,834 issued December 4, 1990 to Xu et al) the Cedar, Sigma-1, the Butterfly and the Monarch, the Intel ipsc, The Connection Machines, the Caltech COSMIC, the N Cube, IBM's RP3, IBM's GF11, the NYU Ultra Computer, the Intel Delta and Touchstone.

Large multiple processors beginning with ILLIAC have been considered supercomputers. Supercomputers with greatest commercial success have been based upon multiple vector processors, represented by the Cray Research Y-MP systems, the IBM 3090, and other manufacturer's machines including those of Amdahl, Hitachi, Fujitsu, and NEC.

Massively Parallel (MP) processors are now thought of as capable of becomming supercomputers. These computer systems aggregate a large number of microprocessors with an interconnection network and program them to operate in parallel. There have been two modes of operation of these computers. Some of these machines have been MIMD mode machines. Some of these machines have been SIMD mode machines. Perhaps the most commercially acclaimed of these machines has been the Connection Machines series 1 and 2 of Thinking Machines, Inc.. These have been essentially SIMD machines. Many of the massively parallel machines have used microprocessors interconnected in parallel to obtain their concurrency or parallel operations capability. Intel microprocessors like i860 have been used by Intel and others. N Cube has made such machines with Intel '386 microprocessors. Other machines have been built with what is called the "transputer" chip. Inmos Transputer IMS T800 is an example. The Inmos Transputer T800 is a 32 bit device with an integral high speed floating point processor.

As an example of the kind of systems that are built, several Inmos Transputer T800 chips each would have 32 communication link inputs and 32 link outputs. Each chip would have a single processor, a small amount of memory, and communication links to the local memory and to an external interface. In addition, in order to build up the system communication link adaptors like IMS C011 and C012 would be connected. In addition switches, like a IMS C004 would be profided to provide, say, a crossbar switch between the 32 link inputs and 32 link outputs to provide point to point connection between additional transputer chips. In addition, there will be special circuitry and interface chips for transputers adapting them to be used for a special purpose tailored to the requirements of a specific device, a graphics or disk controller. The Inmos IMS M212 is a 16 bit process, with on chip memory and communication links. It contains hardware and logic to control disk drives and can be used as a programmable disk controller or as a general purpose interface. In order to use the concurrency (parallel operations) Inmos developed a special language, Occam, for the transputer. Programmers have to describe the network of transputers directly in an Occam program.

Some of these MP machines use parallel processor arrays of processor chips which are interconnected with different topologies. The transputer provides a crossbar network with the addition of IMS C004 chips. Some other systems use a hypercube connection. Others use a bus or mesh to connect the microprocessors and there associated circuitry. Some have been interconnected by circuit switch processors that use

switches as processor addressable networks. Generally, as with the 14 RISC/6000s which were interconected last fall at Lawarence Livermore by wiring the machines together, the processor addressable networks have been considered as coarse-grained multiprocessors.

Some very large machines are being built by Intel and nCube and others to attack what are called "grand challenges" in data processing. However, these computers are very expensive. Recent projected costs are in the order of $30,000,000.00 to $75,000,000.00 (Tera Computer) for computers whose development has been funded by the U.S. Government to attack the "grand challenges". These ."grand challenges" would include such problems as climate modeling, fluid turbulence, pollution dispersion, mapping of the human genome and ocean circulation, quantum chromodynamics, semiconductor and supercomputer modeling, combusion systems, vision and cognition.

Our Massively Parallel Diagonal-Fold Tree Array Processor architecture, which is the subject of this patent, is applicable for modeling high computational parallel data algorithms, for example matrix processing and high connectivity neural networks. To demonstrate the general processing capability of our system an example of matrix multiplication is included.

Problems addressed by our MP Diagonal-Fold Tree Array Processor.

It is a problem for massively parallel array processors to attack adequately the matrix processing problems which exist.

SUMMARY OF THE INVENTION

Our newly developed computer system may be described as a Massively Parallel (MP) Diagonal-Fold Tree Array Processor which operates in a Single Instruction Multiple Data (SIMD) fashion with general purpose application capability. The MP system we prefer will have a N^2 Processor Element (PE) structure in which each PE contains instruction and data storage units, receives instructions and data, and executes instructions. The N^2 PE structure should contain N communicating ALU trees, N Tree Root Processors (TRP), and a mechanism for communicating both instructions and data back to the PEs by means of the communicating ALU trees.

The preferred apparatus which will be described contains N^2 PEs placed in the form of a N by N matrix, with PEs identified by column-row subscripts $PE_{column,row} = PE_{ij}$, that has been folded along the diagonal and made up of Diagonal-PEs and General-PEs.

In our preferred system, the Diagonal-PEs are comprised of single Processing Elements, PE_{ii}, and the General-PEs are comprised of two symmetric Processing Elements, PE_{ij} and PE_{ji}, that are merged together and which are associated with the same PE elements of the N by N PE array prior to folding.

Our new organization of PEs and new PE architecture is described in the best way we know to implement the improvements with an example implementation for matrix multiplication and discussion concerning neural network emulation, matrix addition, and Boolean operations.

These and other improvements are set forth in the following detailed description. However, specifically as to the improvements, advantages and features described herein, reference will be made in the description which follows to the below-described drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIGURE 1 illustrates a vector matrix multiplication operation;
FIGURE 2 illustrates general matrix multiplication;
FIGURE 3 shows a 2 value multiplication structure in two parts, FIGURE 3-A (Diagonal Cell) and FIGURE 3-B (General Cell);
FIGURE 3 illustrates our preferred Processor architecture in two parts, FIGURE 3C (DIAGONAL-PE) and FIGURE 3D (GENERAL-PE); while
FIGURE 4 shows a preferred communicating ALU tree.
FIGURE 5 illustrates a 4^2 PE and 4-Root Tree Processor Massively Parallel Diagonal-Fold Tree Array Processor;
FIGURE 6 illustrates a Processor Element tagged instruction/data format; while
FIGURE 7 illustrates a Processor Element Example Instruction Set.

MATRIX PROCESSING BACKGROUND

A vector matrix multiplication operation utilizing a sum of product calculation especially suited for our preferred MP organization is shown in -- Fig 'MATRX1' unknown -- where there are i columns and j rows.

The input matrix z_i is defined as:

$z_i = Y_1 W_{i1} + Y_2 W_{i2} + \ldots + Y_N W_{iN}$

This is a subset of the general case of matrix multiplications. Consider the three N x N matrices, Y, W, and (Y * W) result matrix z, as shown in -- Fig 'MATRX2' unknown --, with i columns and j rows and a notation of $Y_{(column),(row)}$. It is assumed that a MP Diagonal-Fold Tree Array Processor is available with the following assumed capabilities:

- N Root Tree Processors: each possessing a Y value memory capacity of N Y values and an additional memory capacity for N result values.
- N^2 PEs with the W values to be stored in internal PE registers.
- The Root Tree Processor complex issues all instructions in broadcast mode.

The algorithmic procedure to be described represents only one of many possibilities and is not necessarily the "best" procedure depending upon an application. It is meant to demonstrate the capability of our MP Diagonal-Fold Tree Array Processor. The basic procedure is to have N Root Tree Processors send a row of the Y matrix and a multiplication instruction, with the Auto mode specified, to the PEs which execute the multiplication and send the results to the CATs for summation, providing a row of the result matrix back to the Root Tree Processors for storage. The Root Tree Processors then read out a new row of the Y matrix and send it back to the PEs, continuing generating a row of the result matrix at the output of the CATs and storing the row in memory until all result rows have been calculated. The W value matrix, once initialized in the PEs, remains fixed, internal to the PEs throughout the matrix multiplication operations.

PROCESSOR ELEMENT ARCHITECTURE

Internally, a triangular scalable neural array processor TSNAP structure utilized two types of "cell" structures, the Diagonal-Cells and the General-Cells for the direct emulation of the neural sum of products function and did not address the processing of locally stored data, for example as required by learning algorithms see - Rumelhart 86. The basic multiplier element structures, without programmability are repeated in -- Fig 'SPA1' unknown --A and B, and new "cell" processing structures, with the additional local data manipulation capabilities provided by an instruction set architecture, are shown in and -- Fig 'SPA1' unknown --C and D. The term "element", used in the following discussion, refers to the minimum architected functional unit required TSNAP neural emulation capabilities, namely a weight register, a Y value register, and a multiplier symbolically indicated as $W_{ij} Y_j$. The register terminology of "Y" and $W_{ij} Y_j$ is kept through out the general processor discussion with out limiting the register usage to the neural emulation paradigm. The first "cell", -- Fig 'SPA1' unknown --A, is associated with the diagonal elements, $W_{ii} \times Y_i$, and the second "cell", General-Cell, --Fig 'SPA1' unknown --B, is associated with the rest of the elements $W_{ij} Y_j$ and contains two elements placed in the General-Cell, shown in a top and bottom arrangement. The new form of processor cell, in general referred to as the Processor Element (PE), consists of the Diagonal-PE and the General-PE. The modifications to the basic processing structure, -- Fig 'SPA1' unknown --A and B, are shown in --Fig 'SPA1' unknown --C and D and consist in the addition of a tag compare function, an optional instruction buffer, an instruction (INSTR) register, an expanded register file, Conditional Execution Bits (CEB) in each data register, a data path register, an instruction path bit, selector and distributor control structures, and expanded functions, such as division, square root, etc. that may be application specific in addition to multiplication, as represented by the EXecution Unit (EXU) block. These modifications are required for multiple reasons. First, since a processing element receives both instructions and data from the same source path, namely from the cell attached communicating ALU tree (CAT), a method of differentiating instructions from data must be used. It is assumed that a fixed format is used for the instructions and data allowing a fixed single bit field be used to differentiate instructions from data. Second, by utilizing programmable data path and instruction path registers in conjunction with a front end decoding and distribution mechanism, the destination for a received instruction or data word can be specified. Third, a tag compare function allows further capability in specifying instruction and data destination points. Fourth, since an instruction may specify multiple functions, an instruction register and instruction decoding, operand selection, function selection, destination selection, and execution mechanisms must be provided. Fifth, to provide flexibility in algorithmic capability, a register file with Conditional Execution Bits are provided along with capability of manipulating the CEBs. The execution of an instruction, whose result destination is a local PE register, is conditional based upon the state of the destination register's CEB. The CEB indicates whether a register can be modified or not. The PE programmability is obtained through the instructions which are decoded at a PE's instruction register received from either an optional instruction buffer or with

no instruction buffer from the attached communicating ALU tree that is in a communications mode. Each PE upon receipt of an instruction in the instruction register will execute the operation specified by that instruction. The instruction types include a data/instruction path specification, data movement, arithmetic, and logical instructions. Each PE contains an instruction register for each processing element specifying the source and destination paths and EXU functions; a Diagonal-PE contains one instruction register and the General-PE contains two instruction registers.

The modification to the T-SNAP cells must preserve the functional capabilities provided by the original cells, inorder to support neural emulation as well as other applications requiring similar capabilities. An essential, novel, and general purpose functional capability provided by the T-SNAP multiplier cells, that must be maintained in the new processor cell structure, concerns the emulation of completely connected processors, for example neuron processor as used for completely connected networks such as Hopfield 82 and Hopfield 84. This important function is briefly reviewed using the original T-SNAP cells -- Fig 'SPA1' unknown --A and B.. For example, with a neural network model in an execution mode, implying a multiplication operation in each processing cell, the diagonal cell multiplies its stored weight with its stored Y value and supplies the multiplied result to the attached add tree. In the communications mode for the diagonal cells, a Y value is received from the attached add tree and stored into the Y value register. The "General-Cells" of the structure also generate a weight times Y value and supply the product to their attached add trees. In the communications mode for these "General-Cells", a Y_j value received from the bottom multiplier add tree is stored into the top Y value register and likewise a Y_i value received from the top multiplier add tree will be stored into the bottom Y value register. This switch in storing the Y values is an essential characteristic supporting complete connectivity. For the modified processing cells, -- Fig 'SPA1' unknown --C and D, this path switch is programmable allowing further unique architectural features for processing, as will be described in the Processor Element Instruction Set section of this Chapter. To preserve the internal path switch function of the original T-SNAP cells, the new processor cells require that the data path registers be specified (loaded) in advance of receiving data from a tree. The data path register specifies the destination of the Yj data received from the bottom tree to be the top Yj register and the destination of the Yi data received from the top add tree to be the bottom Yi register thereby preserving the complete connectivity function.

The symbolic summation tree is shown on the left of -- Fig 'TREE1' unknown -- with ALUs at each stage designated by the letter A. The more detailed representation of the communicating ALU tree structure that will be used is shown on the right-hand side of -- Fig 'TREE1' unknown --. Pipeline latches have been left out for more clarity. For specific applications, the ALU function might be as simple as a bit-serial adder or provide more complex programmable functions requiring an instruction set architecture. For the purposes of describing the function execution and communications operations a summation operation may be referred to in this text. The use of the summation function is for simplicity of explaination and not intended to imply a limit to the functionality the communicating ALU tree can provide. In addition, the tree nodes' control mechanism, that determines the nodes operational mode and function, can use separate control lines or tagged tree node instructions. For a single node function such as addition and two operational modes, namely communictions and function execution, a single control line implementation is feasible. If more extended functions are to be supported in a tree node, then not only would additional control mechanisms be required but storage elements may be required in a tree node. In addition, if multiple functions are provided in the tree nodes then a method of synchronistically controlling tree operations must be utilized. If varying function execution timings are to be allowed in each tree node then an asynchronous interfacing method must be provided between the tree stages. For simplicity of implementation that guarantees the synchronization control, a restriction could be enforced that the same operation be specified for each tree stage. In -- Fig 'TREE1' unknown -- three ALU elements are shown in a 2 stage pipelined tree arrangement. The ALU element has a SWitch 1, SW1, block on its output and two SWitch 2s, SW2, blocks bypassing the ALU. The communicating ALU tree can be placed into one of two modes, namely a function execution mode and a communications mode, also termed a bypass mode. A common control signal is used at each ALU element in order to guarantee that all nodes of the tree provide the same mode of operation. One of the functions specified by the tree control signal, an accompanying tag signal or common distributed signal, is the ALU bypass. Both switches, SW1 and SW2, have an on/off control which, when in the "off" state, keeps the switch open, i.e. in a high impedance state and when in the "on" state bypasses the ALU (node function) via a low impedance path. When SW1 is enabled SW2 is disabled and vice versa. In this manner the ALU tree can provide the summation function, for example, in one direction, SW1's on - SW2's off, while essentially acting as a communication path in ALU bypass mode, SW1's off - SW2's on. The ALU tree using 2 to 1 functional elements, such as 2-1 adders, will require $\log_2 N$ stages. Alternatively, the ALU function and communication mode can be implemented with 3-1, 4-1, ..., N-1 functional elements,

such as 3-1, 4-1, ..., N-1 adders, and their bypass switches, utilizing all the same element types or in combination, to produce the specified function. It should be noted that the Communicating ALU, -- Fig "TREE1' unknown --, represents its logical function since, for example, depending upon technology, the SW1's function could be incorporated in the gate devices used in the last internal stage of each ALU element, thereby adding no additional delay to the ALU function. Alternatively, a separate communications tree path could be provided, thereby allowing communications to occur while an ALU function is in progress.

A 4 Root Tree Processor example is shown in -- Fig 'SPA4N' unknown -- which connects the sixteen PEs with four CATs and four Root Tree Processors with a Host interface to provide a complete picture of the machine organization used in the Massively Parallel Diagonal-Fold Tree Array Processor. The CATs are assumed to provide a summation function in ALU exectuion mode. An example of the elements involved in a sum of "W * Y" register products calculation for the third Root Tree Processor RTP_3 is written here and highlighted in -- Fig 'SPA4N' unknown --.

$$RTP_3 = F(W_{3,1}Y_1 + W_{3,2}Y_2 + W_{3,3}Y_3 + W_{3,4}Y_4)$$

The Host interface represent a central control point for the array of PEs allowing the Host to have access to the Root Tree Processors'internal storage possibly containing, for example, the initial parameters W, Y, etc., calculated values, and traced values. There is assumed to be a Root Tree Processor for each communicating/function execution tree and their N attached PEs. Each Root Tree Processor issues instructions and data to the N tree attached PEs through the communications mode of tree operation. Additional functions the Root Tree Processor and Host interface include the following:

1. All processor initializations
2. Starting the system
3. Stoping the system
4. communicating ALU tree control
5. PE instruction and data issuing

In operation, the N^2 PE structure might require an initialization of certain registers. Even though specific registers could be initialized by sending uniquely PE tagged values to individual processors, N^2 operations would be required. An alternative scheme would be to connect the PE registers in a serial daisy chained fashion for LSSD scanning - see Eichelberger 77. LSSD scanning supports testing and register initialization. Each Root Tree Processor and its attached N PEs may have one or up to multiple scan strings depending upon an implementation technology. For example, a separate scan path containing only the "W" registers for each Root Tree Processor can be provided resulting in N "W" register scan paths which can be used for initialization purposes. Other initialization mechanisms are clearly possible and it will be assumed the appropriate method can be chosen during the implementation design process. Parameter values common to a Root Tree Processor's N attached PEs can be loaded through the communicating ALU tree.

PROCESSOR ELEMENT INSTRUCTION SET

An example instruction set providing the previously discussed capability will be reviewed in this section, beginning first with a presentation of an example format for the instructions and data and continuing with an example instruction set description.

A set of decisions must be made to determine an architecture for a processing element. One major set of decisions concerns the format of instructions and data - see Flynn 85 for a review of the universe of format options available and their impact to cost and performance) and the other major set of decisions concerns the functions to be executed by the architecture. Since each Root Tree Processor issues instructions and data to N Processors Elements, a desirable capability for enhanced programming flexibility would be to extend the instruction and data format to include a processor group identification field. Consequently, the instruction format has the requirements to identify processors or groups of processors, separate instructions from data, identify operands, and result destinations. Even though other formats are clearly possible, we choose the example format depicted in -- Fig 'SPA4' unknown --. Both specific Processor Elements and groups of PEs can be identified with the addition of tag bits and a Broadcast, "B", bit added to all communications, and tag and "B" bit compare functions in each PE. A "B" bit is added as the first bit in the field indicating a broadcast instruction/data for B = 1 and a tagged instruction/data for B = 0. A broadcast message/data goes to all N Processor Elements linked to a particular Y_i Root Tree Processor, independent of the TAG. The TAG field must be at least $\log_2 N$ bits long, if specific identification of all Processor Elements belonging to a Root Tree Processor is to be accomplished. Alternatively, groups

of Processor Elements can utilize the same tag value, thereby uniquely identifying groups of PEs. The received tag is bit by bit compared with a stored tag in each PE. After the last tag bit compare is completed, it is known whether the following INSTR/DATA is to be received by that particular PE. A tag match results in a INSTR or Data being received, while a no match situation prevents the reception of a INSTR or Data. A parity bit or error correction bits denoted by a P can also be included in the tag field, as shown in -- Fig 'SPA4' unknown --, for error handling reasons.

The communicated instructions or data also contain a single bit (INSTR) indicating whether the bit string is data or instruction, and for instructions additional fields for specifying an automatic execution mode (AUTO), instruction opcode (INSTR), operand selection (SOURCE1 and SOURCE2), and result destination (DESTINATION). Error correction/detection bit/s (ECC) can be included on both instructions and data for error handling reasons. It is assumed that the instruction and data bit lengths are the same. -- Fig 'SPA5' unknown -- lists the present instruction set functions.

The instruction set may contain, arithmetic operations, for example, add, subtract, multiply, divide, square root, etc., logical operations, for example, AND, OR, EX-OR, Invert, etc., Compare, shift, and data storage movement operations. The instruction set is primarily determined from an application specific perspective.

A fairly standard instruction format is used with the unique addition of the AUTO bit as representing an automatic execution mode. The auto execution mode represents a capability that switches the execution mode of the PEs from an instructions execution only mode to a data dependent mode of execution. The control of the switch from a control flow execution mode to a data flow execution mode is programmable by use of the AUTO bit to engage the data flow mode and a rule that allows the return to control flow instruction execution mode. An instruction with the AUTO bit active is executed first due to normal instruction control flow execution sequencing and then it is executed each time valid data is received in the processing unit. The data flow execution continues until a new instruction is received which stops the previous "AUTO" instruction from executing and begins the execution of the newly received instruction, which may also be another AUTO instruction.

To demonstrate the importance of the AUTO mode for processing, a simple example using the Hopfield neural network will be presented. For this discussion, instruction mnemonics, as presented in the example instruction set architecture of -- Fig 'SPA5' unknown --, are used. Assume the Hopfield neural network model - see Hopfield 84 - is used as an example, for the direct emulation of the network neurons sum of connection weight times connecting neuron output values. Each network update cycle consists of weight times Y value multiplication operations, summation of multiplication results, the generation of the nonlinear sigmoid neuron output Y values, and the communication of the generated Y values to the processing elements. The network updates continue until a network minimum is reached. For simplicity of discussion, assume that network convergence is not tested for on every cycle, but only after some multiple cycles have been executed. For the network emulation using the Processor Elements, an automatic mode can be specified where, instead of requiring the repeated sending of a Multiply instruction to the PEs after each network execution cycle in order to initiate the next network cycle, the automatic mode would begin the next update cycle automatically after receipt of the newly calculated Y values. This automatic mode is initiated by setting the AUTO bit to a "1" in the instruction desired, such as Multiply (MPY) for use in the Hopfield network example, which sets an automatic mode flag in the PEs. The first operation is initiated with the receipt of the instruction with the AUTO bit set to a "1" and the instruction would be repeatedly executed upon receipt of the new updated data continuing until a new instruction is received which terminates the automatic mode, such as receipt of a NOP instruction. A capital A is appended to an instruction mnemonic to indicate that the auto bit is to be set to a "1", for example MPYA.

The source and destination addresses specified in --Fig 'SPA5' unknown -- are relative to the instruction register where the instruction is received. The relative addressing is shown in -- Fig 'SPA1' unknown --D where the top instruction register INSTR TREG relative addresses are shown in columnar fashion, located to the right of the register blocks, while the relative addressing for the bottom instruction register INSTR BREG is shown in columnar fashion, located to the left of the register blocks. For "k" temporary or working registers, it should be noted for example, that the bottom instruction register R2 is the same as the top instruction register R(2+k+1). A bit string received from the ALU tree, if it is an instruction, is serialized into one of the two INSTR registers in each General-cell, as directed by the INSTR PATH BIT, and the single INSTR register of a Diagonal-Cell. A data bit string received from the ALU tree, is serialized to one of the k+4 other registers available in a General-cell and one of the k/2+2 other registers available in a Diagonal-Cell as specified by the DATA PATH register. It is assumed that for a symmetrical structure the Diagonal-Cells contain half the number of instruction and data registers as compared to the General-Cells. In the Diagonal-PEs a source or destination address of R(2+k/2+1) through R(2+k+2) and CR2 are

mapped as follows:

- R(2+k/2+1) → R(2+k/2)
- R(2+k/2+2) → R(2+k/2-1)
- continuing
- R(2+k/2+k/2+2)=R(2+k+2) → R(2+k/2-k/2-1)=R1
- CR2 → CR1

For example, assume a k=2 working registers in the General-Cells and three bit source or destination address then having the General-Cells use all three bits and the Diagonal-Cells use only the 2 lsb bits, the proper mapping can be provided by:

- 000 → CR1
- 001 → R1
- 010 → R2
- 011 → R3
- 100 → CR2
- 101 → R6
- 110 → R5
- 111 → R4

The PATH instruction is treated differently from the other instructions, since it controls the instruction path selection mechanism. The PATH instruction is decoded prior to the distributor logic, -- Fig 'SPA1' unknown -- and the DATA PATH register or INSTR PATH BIT loaded according to destination field. A possible format for the PATH instruction destination field would be to use the first bit position for CR1/2 and the other bits for register path selection, others formats are clearly possible. The PATH instruction must be reissued if a different path is desired. A default path is specified by the architecture for initialization purposes, for example the DATA PATH registers could be initialized to R5, the Y value register to support a completely connected network, and the INSTR PATH BIT set to a "1" indicating the instructions switch path also. All PE data registers are (D = L + 1) bits in length, due to the conditional execution bit in each data register, see -- Fig 'SPA4' unknown -- showing the instruction and data formats. If a CEB is set to a "zero" in an instruction's destination register, that instruction will be treated as a NOP instruction, ie. the destination register's contents will not be changed and "zeros" will be fed to the Add tree. If the CEB is set to a "one" the register's contents can be modified. For example, this bit is used on the W registers to identify the presence or absence of a value since a zero value is not always sufficient to accomplish this. "Zeros" are always fed to the Add tree if the Add tree is not a destination point or a NOP situation is occurring. The CEBs can be set upon initialization through the chips scanning facility, the path instruction, or in a data value received from the tree.

By allowing the instruction and data paths to be programmable, two operational modes for network emulation become possible. In the first mode, termed YIN mode, for all processors, the instruction received from the bottom Adder tree is sent to INSTR BREG (CR1) and the instruction received from the top Adder tree is sent to INSTR TREG (CR1). Then for YIN mode each PE will function as specified in the instruction field. In this fashion, each Root Tree Processor can be specified with a different input PE processing function, common across all PE inputs to that Root Tree Processor. For example, referring to -- Fig 'SPA4N' unknown --, all inputs to Root Tree Processor 4 may be specified with a (W register value * Y value) multiply operation while all inputs to Root Tree Processor 2 may be specified with an Temp. register value * Y value operation. Since all communicating ALU trees are independent of each other, each PE input function can have different execution times in YIN mode for different Root Tree Processors implying that the summation results would occur at different times, which can cause synchronization problems in the interface to the Root Tree Processors, if not handled correctly. YIN mode can be used to asynchronously select a Root Tree Processor and its set of PEs for processing. In the second mode, termed YOUT mode, for all Root Tree Processors, the instruction received from the bottom Adder tree is sent to INSTR TREG (CR2) and the instruction received from the top Adder tree is sent to INSTR BREG (CR2). Consequently, for YOUT mode all Root Tree Processor value outputs will have the same function applied at their input destination PE. In this way each Root Tree Processor can have multiple functions at its input PEs. For example, referring to -- Fig 'SPA4N' unknown --, all Y4 destination PEs may be specified with a (W register value * Y value) multiply operation while all Y2 destination PEs can be specified with a (Temp. register value * Y value) operation. All functions specified at a PE input must execute in the same length of time even though the functions are different. In general, YIN mode and YOUT mode can not be interchanged among the Root Tree Processors within a single model or problem structure as conflicts could result. For the simple Hopfield network emulation example the Root Tree Processors functioning as neurons specified the YOUT mode with all Root Tree Processors issuing the same instruction to all PEs.

Many instructions specify a destination which is local to the individual Processor Element. This local processing can cause synchronization problems if not handled correctly. Instead of proliferating synchronization mechanisms throughout the structure the local processing synchronization problem can be localized to the Root Tree Processors. For example, if no notification of local processing completion is generated from the PEs, a fixed hardware mechanism can be provided at the Root Tree Processor to guarantee safeness of the operations. It is also not desirable to "solve" the problem via means of queues in the Processor Elements as this increases the size of the PE limiting the number which could be placed on a single chip. Rather, the instruction issuing point should be used to resolve and avoid all hazards. Any local processing instruction to the same PE must be separated from the next instruction to that same PE by the specified processor instruction's execution time. For example, if the multiply executed in 2L clocks, a 2L time out must be ensured prior to sending the next instruction. This is necessary so that an instruction buffer register is not required, thereby allowing each instruction to remain constant in a PE during the operation of the function instructioned. Each Root Tree Processor can then be set up with a synchronization mechanism to safely issue instructions to each PE at a maximum rate. Non-local instructions, i.e. those instructions where the destination is the ADD TREE, provide notification of operation completion when the converged tree result reaches the Root Tree Processors. For non-local instructions the Root Tree Processors wait until a result is received before sending a new instruction to the PEs attached to that tree.

As a final note, a compiler would be required to ensure no destination conflicts occur in programs using the described instruction set.

MATRIX PROCESSING EXAMPLE

The following detailed procedure will be followed, see PE example instruction sets: (Note that the PE instructions are indicated by PE-Instruction Mneumonic and don't care states indicated by (x).)

1. CATS placed into communication mode
2. Initialize W matrix into PE registers
3. Each Root Tree Processor memory is initialized as follows:

- Root Tree Processor 1 initialized with Y11, Y12, ..., Y1N.
- Root Tree Processor 2 initialized with Y21, Y22, ..., Y2N.
- .
- .
- .
- Root Tree Processor N initialized with YN1, YN2, ..., YNN.

4. Initialize the Root Tree Processor and PE PATH registers:
   - Set PE INSTR PATH Bit to CR2 indicating YOUT mode
   - Set PE DATA PATH to R2
5. All Root Tree Processors are active
6. Root Tree Processors send the first row of Y values to the PEs.
7. Root Tree Processors send PE-MPYA R1*R2 → ADD TREE, after sending the instruction to the PEs, the Root Tree Processors place the CATs into the summation mode.
8. Root Tree Processors receive the summation results from the CAT roots.
9. Root Tree Processors send the second row of Y values to the PEs.
10. While the PEs and CATs are calculating the next row of the result matrix, the Root Tree Processors can store the first row of the result matrix, in this example to the additional storage capacity assumed present.
11. Since the Auto mode was specified in the PE-MPYA instruction, the PEs upon receipt of the second row Y values will automatically execute a PE-MPY R1*R2 → ADD TREE with CAT results to be received in the Root Tree Processors.

12. The Root Tree Processors send the third row of Y values and stores the second row of the result matrix. The process continues until

13. Root Tree Processors store the last row of the result matrix.

At completion of operation, the original Y and W matrices are intact and the result matrix is located in the Root Tree Processors' additional storage area which can then be further operated upon by the Root Tree Processors or Host system.

Matrix addition and Boolean operations can also be supported by the structure. Assuming matrices of the same form as given in -- Fig 'MATRX2' unknown --, both Y and W matrices can be loaded into the PE array since there are N^2 unique Y and W registers in the structure. Local addition or Boolean operations on the Y and W registers can be done within the structure with the result sent to the temporary registers. At completion of the operation, the original Y and W matrices will remain intact in the structure and the temp regs will contain the result matrix. The result can be scanned out or individually read out from the Processor Element cells or used for further operations (chaining or linking of instructions).

Claims

1. A computer system apparatus for general purpose applications, including matrix processing, is comprised of root tree processors, communicating ALU trees, processing elements (PEs), means for communicating both instructions and data between the root tree processors and the processing elements, and wherein each processor contains instruction and data storage units, receives instructions and data, and execute instructions.

2. The apparatus according to claim 1 further comprising N^2 processing elements, placed in the form of a N by N matrix and identified with a two subscript notation $PE_{column,row}$ that has been folded along the diagonal and made up of diagonal cells and general cells.

3. The apparatus according to claim 2 wherein said diagonal cells, identified as $PE_{i,i}$, are each comprised of a single PE and the general cells, are each comprised of two processing elements, identified as $PE_{i,j}$ and $PE_{j,i}$, that are merged together.

4. The apparatus according to claim 3 wherein the diagonal cells' single PE are each comprised of a tag matching unit, a destination path control mechanism for externally received instructions and data by means of an instruction/data decoding mechanism, a data path storage unit, and a distributor unit, instruction storage units comprised of an instruction buffer that stores zero to X instructions and one instruction storage unit used for instruction decode and operational control, multiple data storage units, a storage unit operand selection mechanism controlled by means of an instruction decoding mechanism, a selector unit, and addressing means relative to the decoded instruction storage unit, a result destination path control mechanism controlled by means of an instruction decoding mechanism and a distributor unit, and a programmable execution unit.

5. The apparatus according to claim 3 wherein the diagonal cells' PEs supply results to and receive instructions and data from an attached communicating ALU tree.

6. The apparatus according to claim 3 wherein the general cells' two PEs, $PE_{i,j}$ and $PE_{j,i}$, merged together are comprised of two tag matching units, a common destination path control mechanism for externally received instructions and data by means of two instruction/data decoding mechanisms, two data path storage units, two instruction path bits, and a common distributor unit, instruction storage units comprised of two instruction buffers that stores zero to X instructions each and two instruction storage units used for instruction decode and operational control, multiple data storage units shared by each of the merged PEs, two storage unit operand selection mechanisms controlled by means of two instruction decoding mechanisms, a common selector unit, and two addressing means relative to the two decoded instruction storage units, two result destination path control mechanisms controlled by means of two instruction decoding mechanisms and a common distributor unit, and two programmable execution units.

7. The apparatus according to claim 3 wherein the general cells' two merged PEs are organized symbolically as a top PE ($PE_{i,j}$) and a bottom PE ($PE_{j,i}$) each of said top and bottom PEs supplying a result to and receiving instructions and data from an attached communicating ALU tree.

8. The apparatus according to claim 1 wherein the PEs' data storage units contain conditional execution bits with one bit per data storage unit which said bit controls the use of the data and whether the data may be overwritten.

9. The apparatus according to claim 1 wherein the root tree processor provides function execution upon data supplied from the communicating ALU trees in a function execution mode and supplies instructions/data to the communicating ALU trees in a communications mode.

10. The apparatus according to claim 1 wherein binary communicating ALU trees contain $\log_2 N$ 2 to 1 communicating ALU stages.

11. The apparatus according to claim 10 wherein each stage in the communicating ALU trees contain 2 to 1 communicating ALUs comprised of a 2 to 1 ALU, an ALU bypass path for the purposes of communicating values in an opposite direction than that used for the ALU execution, and means for switching between the ALU function and the communication's path.

12. The apparatus according to claim 1 wherein the communicating ALU trees each connect to an additional ALU stage wherein an external input value is processed with the output of the communicating ALU tree and said additional ALU stage provides results to the root tree processors.

13. The apparatus according to claim 1 wherein the root tree processors and their Host computer interface provides the following functions:
- communicating ALU tree control
- PE initializations
- PE instruction issuing
- algorithmic data calculations
- PE data issuing
- synchronously starting the PEs into execution mode
- synchronously stopping the PEs

14. The root tree processor controlling apparatus of claim 13 contains a multiple storage arrays corresponding to the PEs storage units supporting initialization procedures, result storage, and tracing operations.

15. The apparatus according to claim 1 wherein there are N^2 PEs, N communicating ALU trees, and N root tree processors for a N array structure.

16. The apparatus according to claim 15 wherein each communicating ALU tree connects to N PEs at the leaf nodes of the tree and one root tree processor which connects to the root of the tree providing results to a Host interface and where said communicating ALU trees, PEs, and root tree processors constituting the N array structure have:
- means for inputting data values to each PE,
- means for communicating tagged instructions and data to the PEs from the root tree processing units,
- means for controlling the destination of instructions and data in each PE,
- means for the execution of the received instructions in each PE,
- means for the execution of a previously received instruction when, in an auto mode, data is received to be used in the next operation,
- means for operand selection and destination path control allowing results to stay locally in each PE or to be sent to the attached communicating ALU tree,
- means for the converged function execution of values received from the multiple PEs,
- means for the inputting of external data values to each root tree processor,
- means for the generation of new instructions and data.

17. The apparatus according to claim 16 wherein the means for inputting data values to each PE comprises a host interface controlling mechanism in the form of a root tree processor and its programmable processor controlling apparatus which has access to each data value storage unit in each PE.

18. The apparatus according to claim 16 wherein the means for communicating tagged instructions and data to the PEs from the root tree processor is by means of the communicating ALU trees acting in a communications mode and tag matching units in each PE wherein the tag comprises a broadcast bit and a tag address field.

19. The apparatus according to claim 16 wherein the means for controlling the destination of instructions and data in each PE is, for instructions, by an instruction decoding mechanism, an instruction path bit, and distributor logic in the general cells and by an instruction decoding mechanism, register mapping logic whereby general cell specified registers are mapped to diagonal cell registers, and distributor logic in the diagonal cells and, for data, by a data decoding mechanism and a data path storage unit in both the diagonal cells and the general cells.

20. The apparatus according to claim 19 wherein in one mode of operation of the general cells, termed the YIN mode, the data path storage units and the instruction path bits are set up such that instructions received from the top communicating ALU tree are directed to the top PE's instruction storage unit and instructions received from the bottom communicating ALU tree are directed to the bottom PE's instruction storage unit and data received from the top communicating ALU tree are directed to the top PE's specified data storage unit and data received from the bottom communicating ALU tree are directed to the bottom PE's specified data storage unit.

21. The apparatus according to claim 19 wherein in a second mode of operation of the general cells, termed the YOUT mode, the data path storage units and the instruction path bits are set up such that instructions received from the top communicating ALU tree are directed to the bottom PE's instruction storage unit and instructions received from the bottom communicating ALU tree are directed to the top PE's instruction storage unit and data received from the top communicating ALU tree are directed to the bottom PE's specified data storage unit and data received from the bottom communicating ALU tree are directed to the top PE's specified data storage unit.

22. The apparatus according to claim 16 wherein the means for the execution of the received instructions in each PE is through a programmable execution unit responding to NOP, PATH, Arithmetic, Logical, shift, compare, and data storage movement instructions containing specifications of an auto operation mode, the source operands, the result destination, and specification of immediate data.

23. The apparatus according to claim 16 wherein the means for the execution of a previously received instruction when, in an auto mode, data is received to be used in the next operation is by means of an auto mode flag as set by a received instruction having the capability of setting the auto mode and by receipt of valid data that is sent after a synchronizing mechanism ensures no conflicts with use of the communicting ALU tree, wherein said synchronizing mechanism comprises a time delay control or use of the communicating ALU tree or alternate signaling means for communicting PE execution status.

24. The apparatus according to claim 16 wherein the means for operand selection and destination path control allowing results to stay locally in each PE or to be sent to the attached communicating ALU tree is by means of an instruction decoding mechanism and distributor logic with no operation completion indication given under a time delay control synchronizing mechanism or by communicating PE execution status through the communicating ALU tree or alternate signaling means.

25. The apparatus according to claim 16 wherein the means for the converged function execution of values received from the multiple PEs is through the attached communicating ALU trees acting in a function execution mode.

26. The apparatus according to claim 16 wherein the means for the inputting of external input values to each root tree processor is through an externally applied input to a final summation stage located at the output of the communicating ALU tree.

27. The apparatus according to claim 16 wherein the means for the generation of new instructions and data is by means of the root tree processors and a programmable controlling apparatus which interfaces to an attached host computer and to the N communicating ALU trees.

28. The programmable processor controlling apparatus of claim 27 wherein a time out state machine controlling mechanism is used on the instruction and data issuing mechanism to avoid hazards on the structure.

29. The apparatus of claim 16 wherein the data are in a bit-serial format which for the data is, in the order the bits are received into a diagonal or general cell, first a broadcast bit, next a tag field, next an error handling bit/s, continuing with an instruction bit set to an inactive state to indicate data, a spare bit, a data field, and ending in error handling bit/s.

30. The apparatus of claim 16 wherein the instructions are in a bit-serial format which for instructions is, in the order the bits are received into a diagonal or general cell, first a broadcast bit, next a tag field, next an error handling bit/s, continuing with an instruction bit set to an active state to indicate an instruction, an auto bit, a instruction field indicating the instruction type, a source-1 field indicating the first operand, a source-2 field indicating the second operand, a destination field indicating the destination of results, an immediate data field, and ending in error handling bit/s.

31. The apparatus of claim 16 wherein there is means provided for sequentially performing matrix multiplications of two N by N matrices, one termed a W matrix and the other termed the Y matrix, where the multiplication creates a third N by N matrix, termed the z matrix, and registers are used for storage units, enabling a process when MPY indicates a multiply instruction, a destination of ALU TREE sends results to the attached communicating ALU tree, the root tree processors includes the Host interfacing function and where the process steps include: a) load W matrix (assuming N W values per root tree processor) b) load first Y row by communicating Y values through the communicating ALU trees c) MPYA R1*R2 → ALU TREE (Where the ALU tree has been initialized for the summation process.) d) calculate first row of result z matrix - multiply Y & W registers followed by summation tree e) store the N z values in the root tree processors f) communicate second Y row through the communicating ALU trees g) when the new Y values have been received, calculate second row of the result z matrix - multiply Y & W registers followed by summation tree h) store the N z values in the root tree processors i) continue with row calculations until j) communicate N^{th} Y row k) when the new Y values have been received, calculate N^{th} row of result z matrix - multiply Y & W registers followed by summation tree 1) store final row of result z matrix in the root tree processors.

32. The apparatus of claim 16 wherein there is means provided for sequentially performing matrix addition of two N by N matrices, one termed a W matrix and the other termed the Y matrix, where the addition creates a third N by N matrix, termed the z matrix, stored internally to the PEs in the temporary storage units, then assuming both Y and W matrices are initialized or in place due to previous calculations and there are N^2 unique Y and W storage units in the structure, the system is enabled to perform the local addition on the Y and W storage units, which addition is done within the PEs with the result sent to the PEs' temporary storage units that after completion of the addition the original Y and W matrices will remain intact in the structure and the temporary storage units will contain the addition result matrix that can be read out or used for further operations.

33. The apparatus of claim 16 wherein there is means provided for sequentially performing matrix Boolean operations on two N by N matrices, one termed a W matrix and the other termed the Y matrix, where the Boolean operation creates a third N by N matrix, termed the z matrix, stored internally to the PEs in the temporary storage units, then assuming both Y and W matrices are initialized or in place due to previous calculations and there are N^2 unique Y and W storage units in the structure, the system is enabled to perform the local Boolean operation on the Y and W storage units, which Boolean operation is done within the PEs with the result sent to the PEs' temporary storage units that after completion of the Boolean operation the original Y and W matrices will remain intact in the structure and the temporary storage units will contain the Boolean operation result matrix which can be read out or used for further operations.

$$\begin{bmatrix} Y1 & Y2 & \cdots & YN \end{bmatrix} * \begin{bmatrix} W11 & W21 & \cdots & WN1 \\ W12 & W22 & \cdots & WN2 \\ \vdots & \vdots & & \vdots \\ W1N & W2N & \cdots & WNN \end{bmatrix} = \begin{bmatrix} z1 & z2 & \cdots & zN \end{bmatrix}$$

FIG.1

$$\begin{bmatrix} Y11 & Y21 & \cdots & YN1 \\ Y12 & Y22 & \cdots & YN2 \\ \vdots & \vdots & & \vdots \\ Y1N & Y2N & \cdots & YNN \end{bmatrix} * \begin{bmatrix} W11 & W21 & \cdots & WN1 \\ W12 & W22 & \cdots & WN2 \\ \vdots & \vdots & & \vdots \\ W1N & W2N & \cdots & WNN \end{bmatrix} = \begin{bmatrix} z11 & z21 & \cdots & zN1 \\ z12 & z22 & \cdots & zN2 \\ \vdots & \vdots & & \vdots \\ z1N & z2N & \cdots & zNN \end{bmatrix}$$

FIG.2

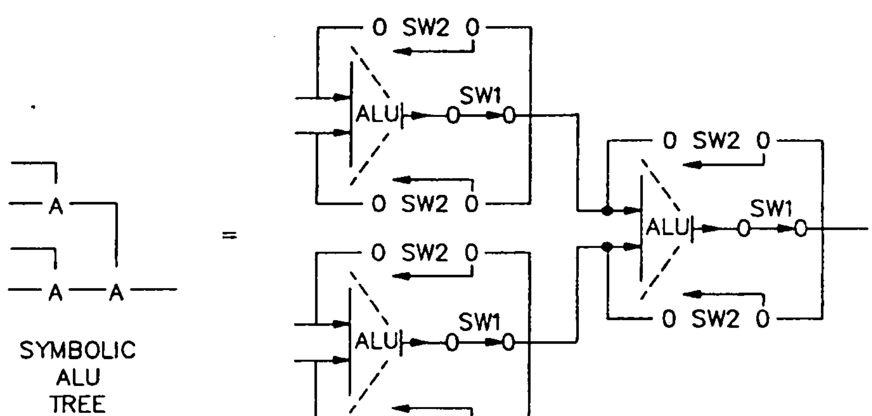
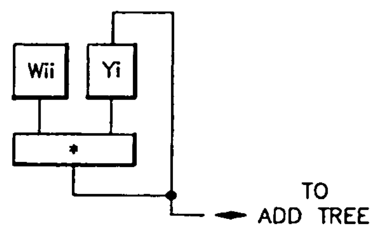


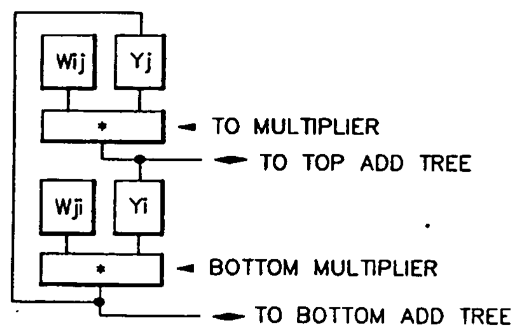
FIG.4



DIAGONAL "CELL"

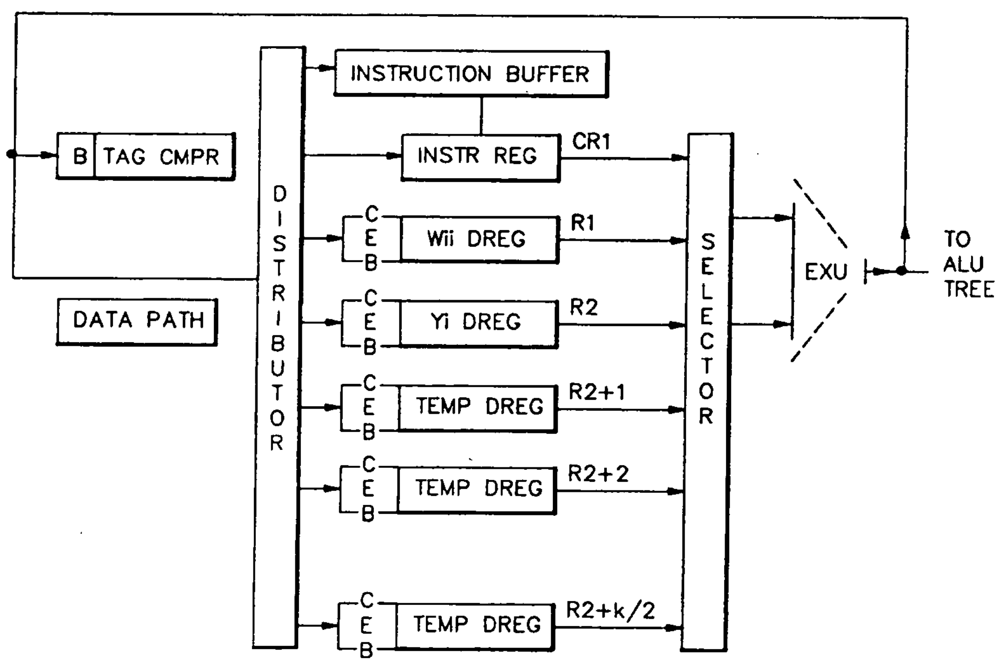
T-SNAP ORIGINAL
WEIGHT/Y VALUE
MULTIPLICATION STRUCTURE

FIG.3A



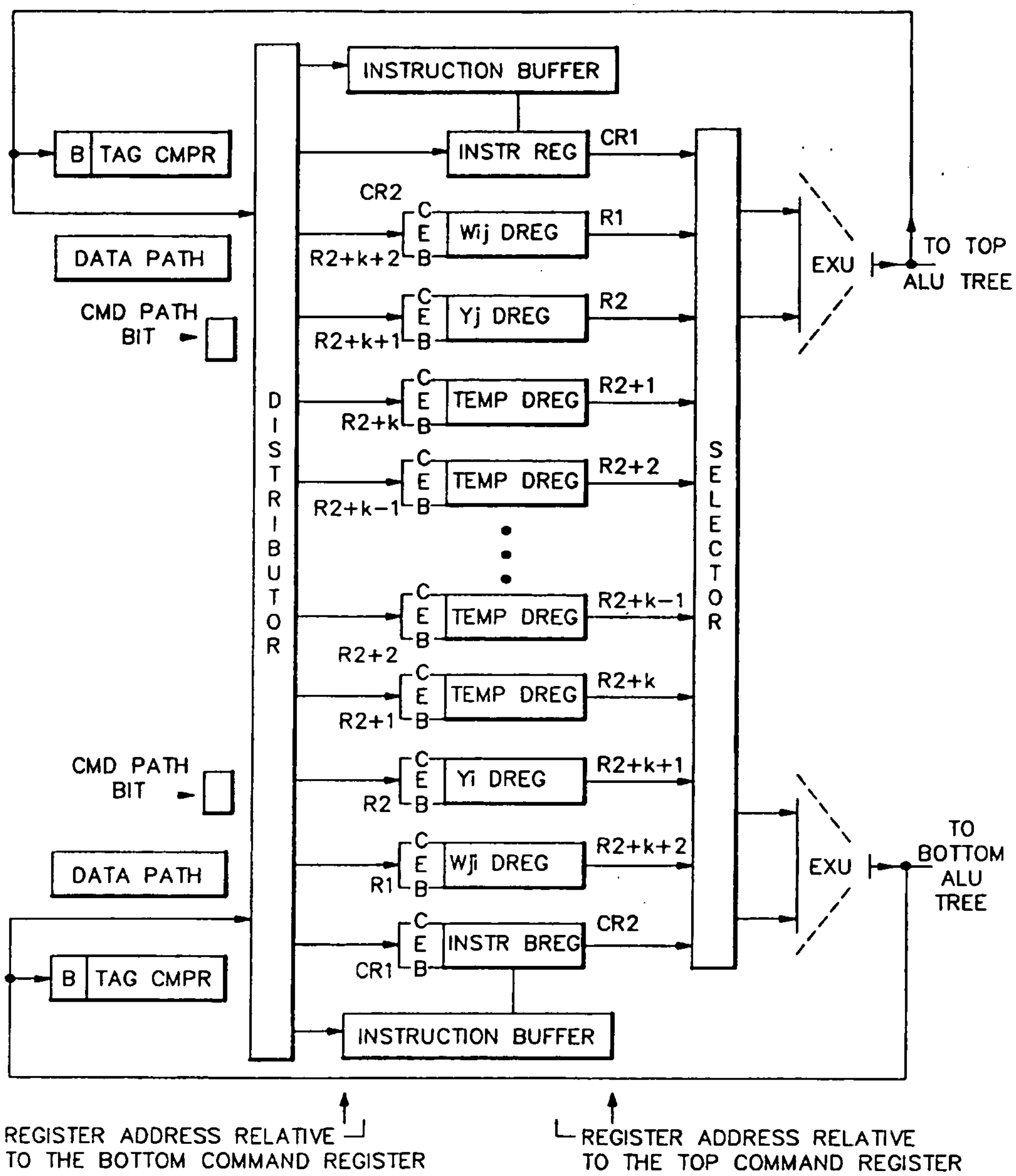
G – "CELLS"

FIG.3B



DREG = DATA REGISTER "K" TEMPORARY REGISTERS

FIG.3C



G-GENERAL PROCESSOR

FIG.3D

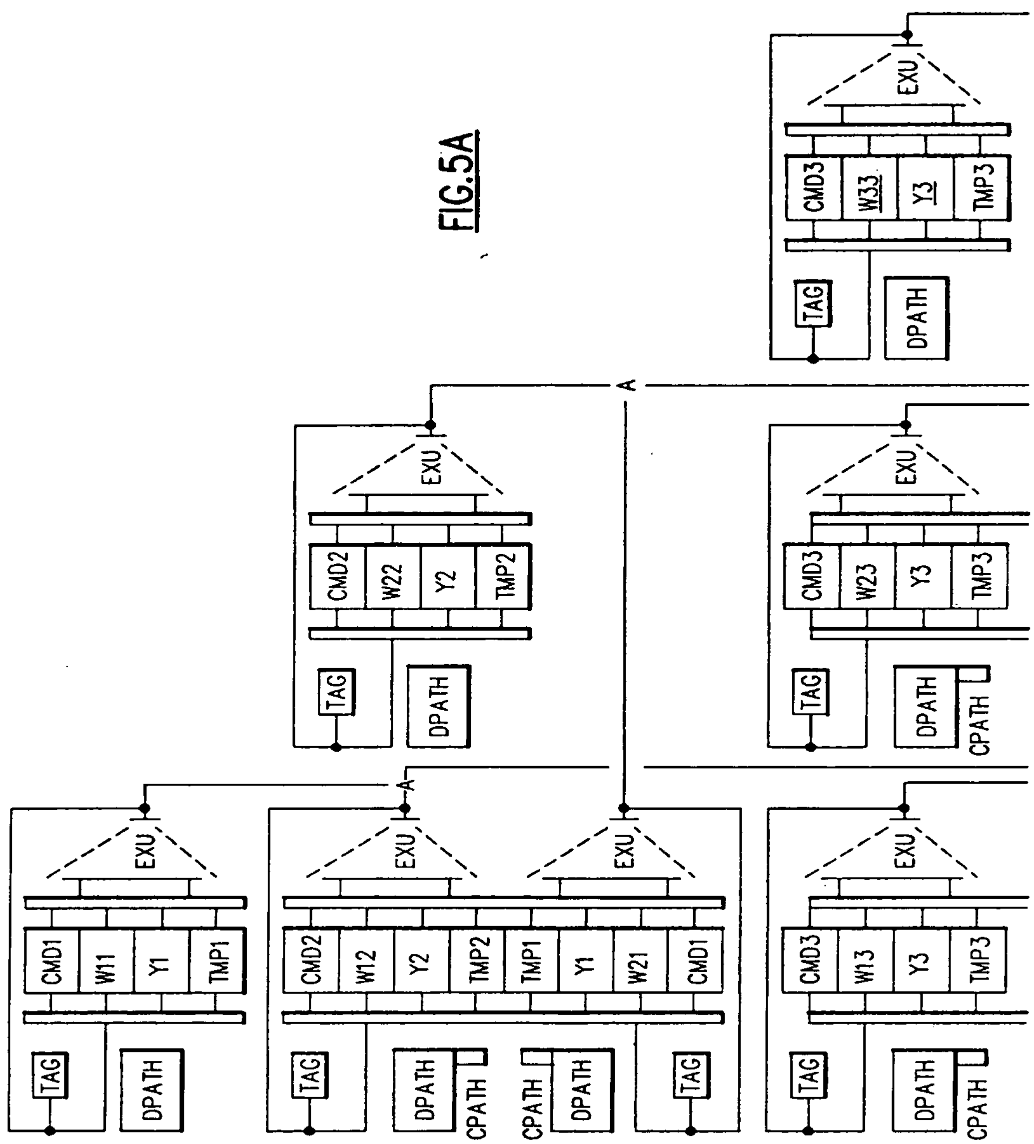

EXU
CMD3
W33
Y3
TMP3
TAG
DPATH
A
CMD2
W22
Y2
TMP2
CMD3
W23
Y3
TMP3
CPATH
CMD1
W11
Y1
TMP1
CMD2
W12
Y2
TMP2
TMP1
Y1
W21
CMD1
CMD3
W13
Y3
TMP3

FIG.5A

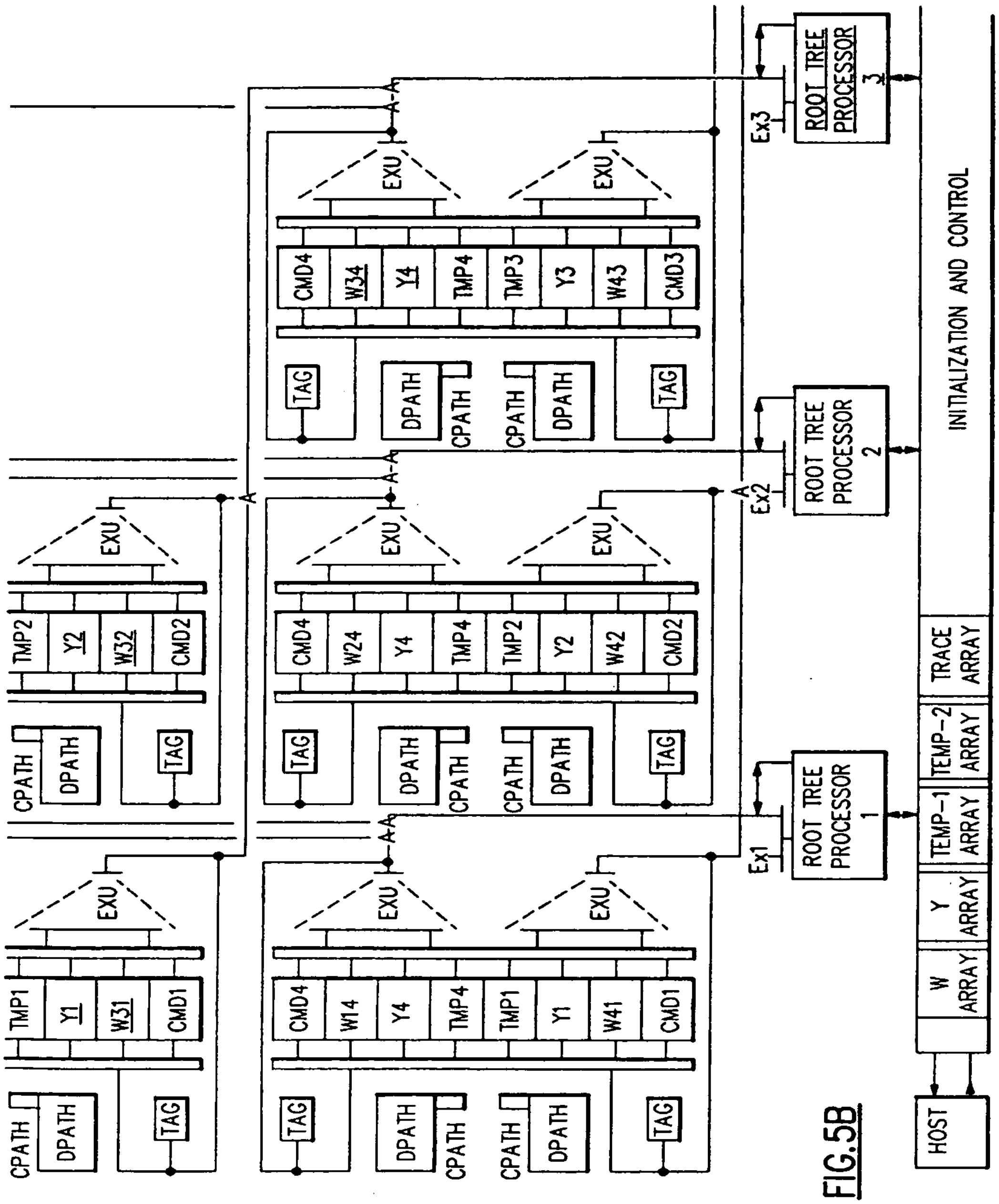

ROOT TREE PROCESSOR 1
ROOT TREE PROCESSOR 2
ROOT TREE PROCESSOR 3
INITIALIZATION AND CONTROL
HOST
W ARRAY
Y ARRAY
TEMP-1 ARRAY
TEMP-2 ARRAY
TRACE ARRAY
Ex1
Ex2
Ex3
EXU
TAG
DPATH
CPATH
CMD1
CMD2
CMD3
CMD4
W14
W24
W34
W31
W32
W41
W42
W43
Y1
Y2
Y3
Y4
TMP1
TMP2
TMP3
TMP4
A
A-A

FIG.5B

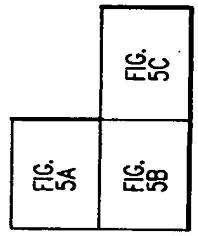
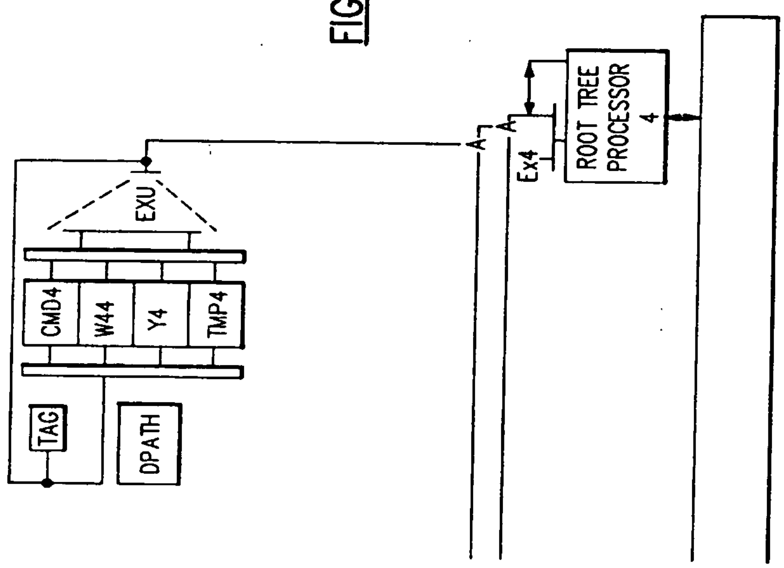


FIG. 5A
FIG. 5B
FIG. 5C

FIG.5

FIG.5C


EXU
CMD4
W44
Y4
TMP4
TAG
DPATH
A
A
Ex4
ROOT TREE PROCESSOR 4

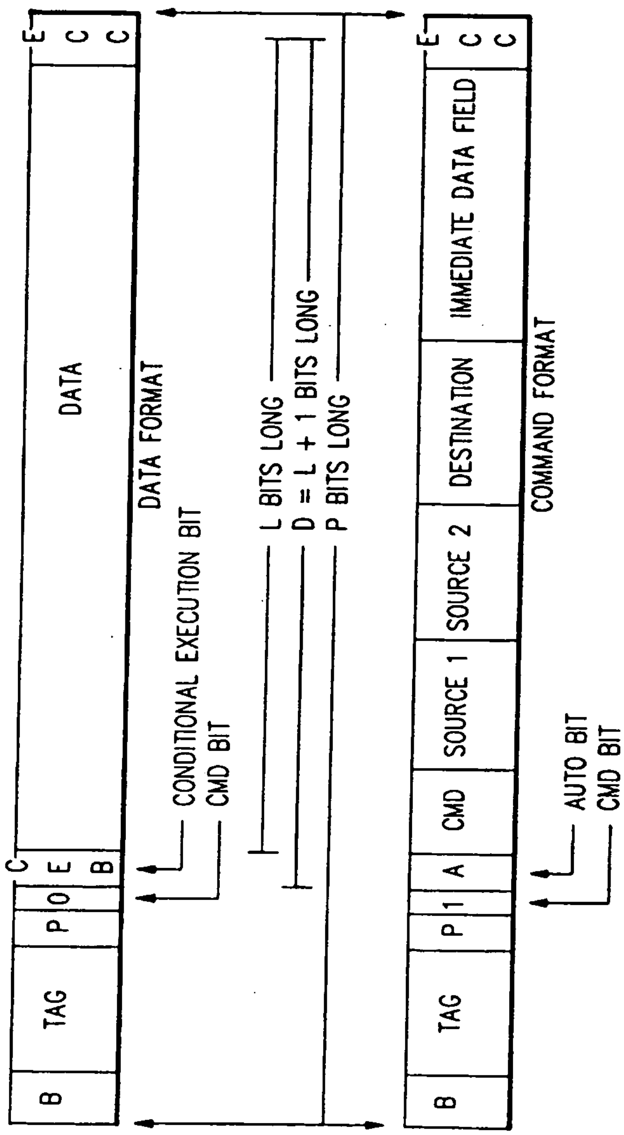

B
TAG
P
0
E
B
C
DATA
E
C
C
DATA FORMAT
CONDITIONAL EXECUTION BIT
CMD BIT
L BITS LONG
D = L + 1 BITS LONG
P BITS LONG
B
TAG
P
1
A
CMD
SOURCE 1
SOURCE 2
DESTINATION
IMMEDIATE DATA FIELD
E
C
C
COMMAND FORMAT
AUTO BIT
CMD BIT

FIG.6

INSTRUCTION	*AUTO*	SOURCE 1	SOURCE 2	DESTINATION
PATH	N.U.	N.U.	N.U.	R1,R2,R3,R4, R5,R6 CR1,CR2
NOP	N.U.	N.U.	N.U.	N.U.
ARITHMETIC (ADD,MPY,DIV, SQRT,etc.)	0=NO 1=AUTO	R1,R2,R3,R4, R5,R6,IMD1, IMD2	R1,R2,R3,R4, R5,R6,IMD1, IMD2	R1,R2,R3,R4, R5,R6, ALU TREE
LOGICAL (AND,OR,EXOR, etc.)	0=NO 1=AUTO	R1,R2,R3,R4, R5,R6,IMD1, IMD2, CEB VECTOR	R1,R2,R3,R4, R5,R6,IMD1, IMD2, CMP FLAGS	R1,R2,R3,R4, R5,R6, ALU TREE CEB VECTOR
INV	0=NO 1=AUTO	R1,R2,R3,R4, R5,R6,IMD1, IMD2, CEB VECTOR	N.U.	R1,R2,R3,R4, R5,R6, ALU TREE CEB VECTOR
CMPR	0=NO 1=AUTO	R1,R2,R3,R4, R5,R6,IMD1, IMD2	R1,R2,R3,R4, R5,R6,IMD1, IMD2	LT, GT, EQ FLAGS
SHIFT	0=NO 1=AUTO	R1,R2,R3,R4, R5,R6,IMD1, IMD2 IMD2	N.U.	N.U.
SENDREG	0=NO 1=AUTO	R1,R2,R3,R4, R5,R6,IMD1, CEB VECTOR, CMP FLAGS	N.U.	ALU TREE

AUTO = 1 → AUTOMATIC REPEAT OF FUNCTION AFTER RECEIPT OF

FIG.7A

IMMED.DATA		COMMENTS
C E B	NOT USED (N.U.)	IF DESTINATION IS CR1 SET THE CMD PATH BIT TO A 0, IF IT IS CR2 SET THE CMD PATH BIT TO A 1. (CEB FIELD NOT USED) ELSE SET THE DATA PATH REGISTER TO THE DESTINATION ADDRESS AND THE DESTINATION REGISTER'S CEB AS SPECIFIED.
N.U.		NO OPERATION
DATA		IMD1/2 = CMD REG 1/2 IMMEDIATE DATA
DATA		CEB VECTOR = (CEB1,CEB2, ..., CEB6) WHERE CEBx = CEB BIT FOR REGISTER Rx
DATA		
DATA		LT = SOURCE-1 SOURCE-2 GT = SOURCE-1 SOURCE-2 EQ = SOURCE-1 = SOURCE-2
SHIFT TYPE & SHIFT AMOUNT		THE FIRST PART OF THE IMMEDIATE DATA SPECIFIES TYPE OF SHIFT OPERATIONS, eg. WITH OR WITHOUT WRAPAROUND. THE SECOND PART SPECIFIES THE NUMBER OF BIT SHIFTS.
N.U.		IF SOURCE-1 = CEB VECTOR THE SIX CEB BITS ARE PACKED INTO THE MSB BITS OF THE RESPONSE. IF SOURCE-1 = CMP FLAGS THE THREE FLAG BITS ARE PACKED INTO THE MSB BITS OF THE RESPONSE.

UPDATED DATA FROM SOURCE EXTERNAL TO PROCESSOR ELEMENT

FIG. 7A	FIG. 7B

FIG.7

FIG.7B